OS MMXX

MIT ;)

https://pdos.csail.mit.edu/6.828

prednaska podla LEC3 a kusok LEC4

Struktura prednasky

Uvod do OS

Start prveho procesu xv6

Izolacia procesov

- Systemove volania
- Virtualna pamat
  - Segmentacia
  - Strankovanie

Opakovanie

- Naco sluzi OS?
- Obrazok

Opakovanie

- Ciele OS

1. Umoznit beh viacerym aplikaciam (sucasne)
2. Izolovat aplikacie
3. Umoznit aplikaciam komunikovat
4. Efektivne vyuzivat hardver

Mozu bezat app bez OS?

- Obrazok
- Aplikacie priamo interaguju s hw
  - CPU
  - RAM
  - HDD
  - …

Mozu bezat app bez OS?

- Problem multiplexingu

- Aplikacia sama sa musi vzdat CPU
  - Ak to programator zabudne urobit, ina app sa k CPU nedostane
  - Ak sa app dostane do nekonecneho cyklu, ina app sa uz k CPU nedostane
  - Nie je mozne ukoncit beh inej aplikacie (kill)
- Tento pristup sa pouziva pri OS realneho casu (kooperativne planovanie procesov)

Mozu bezat app bez OS?

- Problem vzajomneho pristupu do pamate

- Nejestvuje izolacia, vsetky aplikacie maju priamy pristup do pamate
- Aplikacia moze prepisat udaje inej aplikacii
- Aplikacia moze prepisat kod zdielanej kniznice pre vsetky aplikacie
- Kazda aplikacia ma pristup ku vsetkym udajom inych aplikacii

Rozhranie UNIX-like OS

- OS vytvara abstrakciu hardveru

- Procesy ←→ jadra CPU (fork)
- Pamat ←→ fyzicka pamat (exec)
- Subory ←→ diskove bloky (open, read, …)
- Rury ←→ zdielana fyzicka pamat (pipe)
- …

Rozhranie UNIX-like OS

- Procesy ←→ jadra CPU (fork)
  - OS obsadzuje jadra pre beh procesov (uklada a obnovuje registre)
  - OS vynucuje vymenu procesov na CPU
- Pamat ←→ fyzicka pamat (exec)
  - Kazdy proces ma svoju “vlastnu” pamat (obrazok)
  - OS rozhoduje, kam sa do ramky umiestni app

Rozhranie UNIX-like OS

- Subory ←→ diskove bloky (open, read, …)
  - OS poskytuje pohodlne pouzivanie mien suborov a adresare
  - OS moze umoznit zdielanie suborov medzi procesmi/uzivatelmi systemu

- Rury ←→ zdielana fyzicka pamat (pipe)
  - OS riadi synchronizaciu prenosu (ak je rura prazdna, citatel musi cakat) (obrazok)
  - OS moze kedykolvek ukoncit cinnost citatela ci zapisovatela

Rola OS

- OS musi byt defenzivny
- Aplikacia by nemala byt schopna sposobit pad OS
- Aplikacia by nemala byt schopna prerazit barieru izolacie a zasahovat priamo do inej aplikacie

Ako?

- Kedze aj OS aj aplikacia su sw, vacsinou sa to robi pomocou hw

1. Hw podpora roznych urovni vykonavania instrukcii na CPU (user / kernel mod)

2. Hw podpora virtualnej pamate (vytvorenie zdania “vlastnej” pamate pre proces)

User / Kernel mod

- Uz v minulej prednaske sme spominali, ze kernel mod znamena moznost vykonavania vsetkych instrukcii, aj tzv. privilegovanych
  - Napr. nastavenie modu procesora
  - Alebo priamy pristup k hw
- V user mode nie je mozne privilegovane instrukcie vykonat
  - Pri pokuse sa vyvola hw vynimka, ktoru ma moznost obsluzit programovy kod v kernel rezime
- OS bezi v kernel mode, procesy v user mode

Virtualna pamat

- SW pouziva adresy (ukazatele, pointre)
  - Nazyvame ich virtualne
  - Plati to ako pre OS aj pre aplikacie pouzivatela
- HW poskytuje tzv. tabulky stranok, pomocou ktorych sa robi (hardverovo!) preklad virtualnej adresy na fyzicku (do ramky)
- OS nastavuje tieto tabulky pre proces tak, aby nemal pristup k datam ineho procesu
- Tabulky urcuju, do ktorej fyzickej pamate ma ta-ktora aplikacia pristup

Komunikacia app s OS

- Ak sa vyuzije hw na takto silnu izolaciu, je potrebne vymysliet mechanizmus, pomocou ktoreho moze aplikacie **kontrolovanym sposobom** pristupit k hw (alebo k udajom inej aplikacie)
- V podstate sa tento problem redukuje na kontrolovany prechod z uzivatelskeho rezimu procesora do kernel modu procesora
- Na architekture RISCV sa to deje pomocou instrukcie ecall <n>

ecall <n>

- Vyvolanie sluzby OS presne definovanym sposobom
- Procesy nemaju pristup k funkciam jadra OS priamo!

- Systemove volania nie su “klasicke” funkcie
  - user/forktest.asm hladaj fork
  - user/usys.pl →user/usys.S
  - hw prepnutie do kernel modu; OS urcuje, kde sa po prepnuti zacne vykonavat kod jadra OS

Struktura xv6

- Monoliticky kernel
  - Rozhranie user/kernel space: systemove volania
- Zdrojove kody su usporiadane modularne
  - user/ → aplikacie v user mode
  - kernel/ → kod v kernel mode
- Samotny kernel pozostava zo samostatnych casti
  - Vid kernel/defs.h (proc, fs, …)
- Kod je vynikajuco dokumentovany
  - Moznost pochopit ho aj bez citania knihy o xv6

Pouzitie xv6

- Subor zostavenia Makefile riadi
  - Vytvorenie programu jadra (priecinok kernel/)
  - Vytvorenie uzivatelskych programov (priecinok user/)
  - Vytvorenie disku (priecinok mkfs/)

- Prikaz `make qemu`
  - Spusta xv6 vo virtualizacnom nastroji qemu
  - Qemu emuluje pocitac architektury RISC-V

RISC-V doska

- Ide o realny hw → moznost skutocneho spustenia xv6!
- Velmi jednoducha zakladna doska (bez grafickeho vystupu)
  - CPU ma 4 jadra
  - RAM 128MB
  - Podpora preruseni
  - Podpora UART (seriova konzola/klavesnica)
  - Podpora sietovej karty e1000 (cez zbernicu PCIe)
- Qemu emuluje presne tuto konfiguraciu

Preco qemu a nie hw?

- Museli by ste si ho zakupit (a teraz to moze z Ciny trvat dost dlho!)

- Pohodlnejsie je vyuzivat sluzby qemu, nakolko
  - Qemu emuluje viacero implementacii RISC-V
  - Xv6 vyuziva pocitac “virt” (https://github.com/riscv/riscv-qemu/wiki)
  - Tato implementacia je dost blizka hw doske SiFive, ale navyse ma podporu pre rozhranie VirtIO (https://www.sifive.com/boards)

Co znamena emulacia?

- Qemu je sw (program), ktory implementuje RISC-V procesor

```
for (;;) {
    read next instruction
    decode instruction
    execute instruction (updating processor state)
}
```

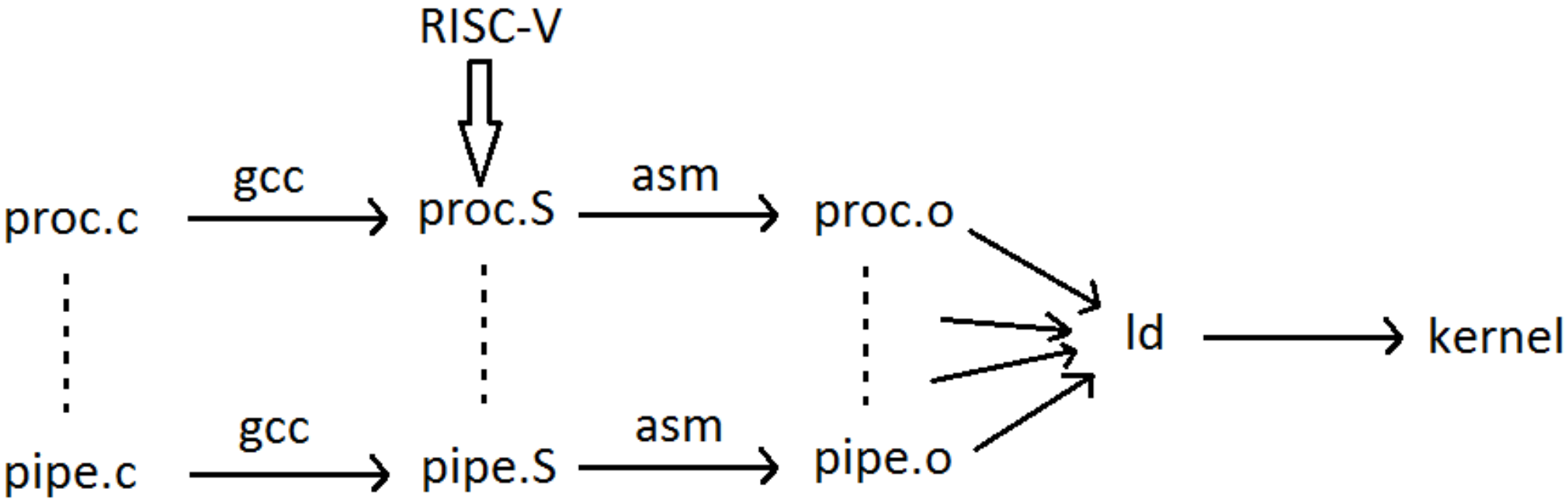
Start systemu xv6

- Vyuzijeme gdb
- Spustime xv6 s jednym jadrom (nie ako je prednastavena hodnota 4), aby sme mohli sledovat jedno vlakno v gdb

```
$ make CPUS=1 qemu-gdb
```

- Qemu zacne vykonavat kod xv6, ktory sa nachadza v kernel/entry.S
  - Vid kernel/kernel.ld symbol _entry (riadok 2)
  - Co je kernel.ld?

Zostavenie xv6



prevzate a upravene z https://pdos.csail.mit.edu/6.828/2020/lec/l-os-boards.pdf

Makefile je nastaveny tak, aby vytvoril asm subory (vid napr. kernel.asm)

Start systemu xv6

- b _entry
  - Porovnajme instrukcie so suborom
  - Info reg
- b main
  - Next next next next …
- step do funkcie userinit()
  - Next cez funkciu
  - Vid proc.h
  - Step do allocproc()
  - Vid initcode.S/initcode.asm a `od -t xC initcode`

Start systemu xv6

- b forkret
  - Next po usertrapret()
- b syscall
  - print num
  - Step do syscalls[num]
  - Nachadzame sa v exec “/init”
- symbol-file user/init.o
  - b main
  - continue

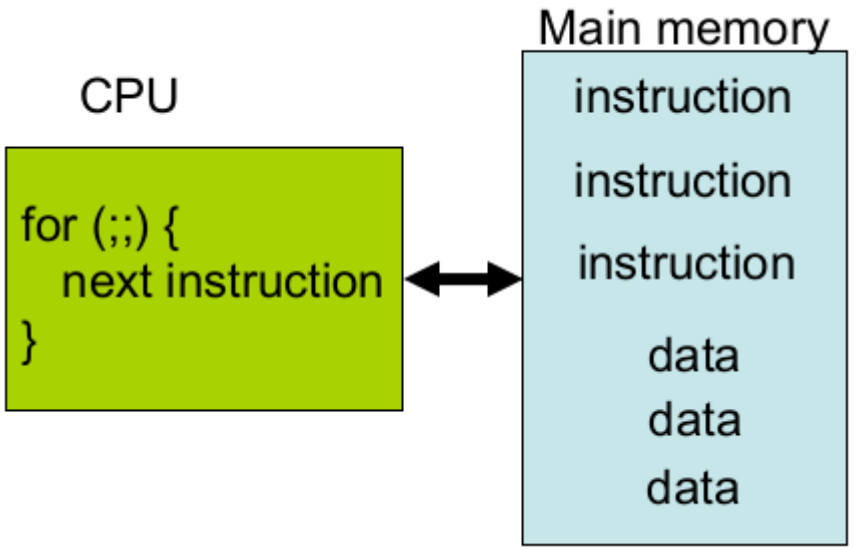
Nejake otazky?

Von Neumann architektura

- CPU
- Memory
- I/O
- Bus

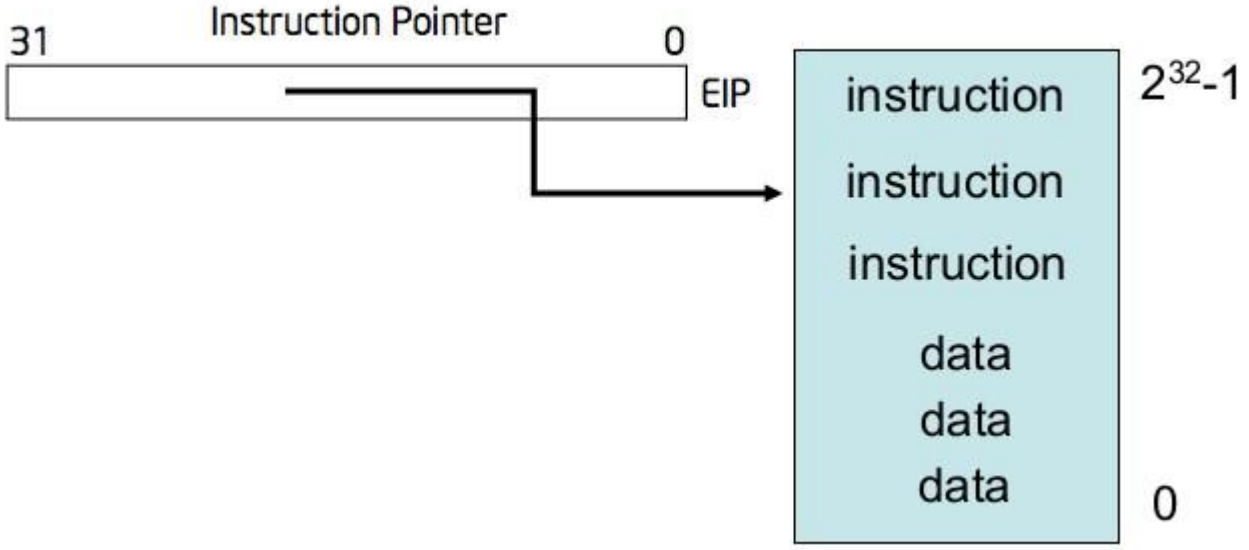
Vykonavanie programu

- Pamat
  - Instrukcie
  - Udaje
- CPU
  - Interpretacia
  - Manipulacia

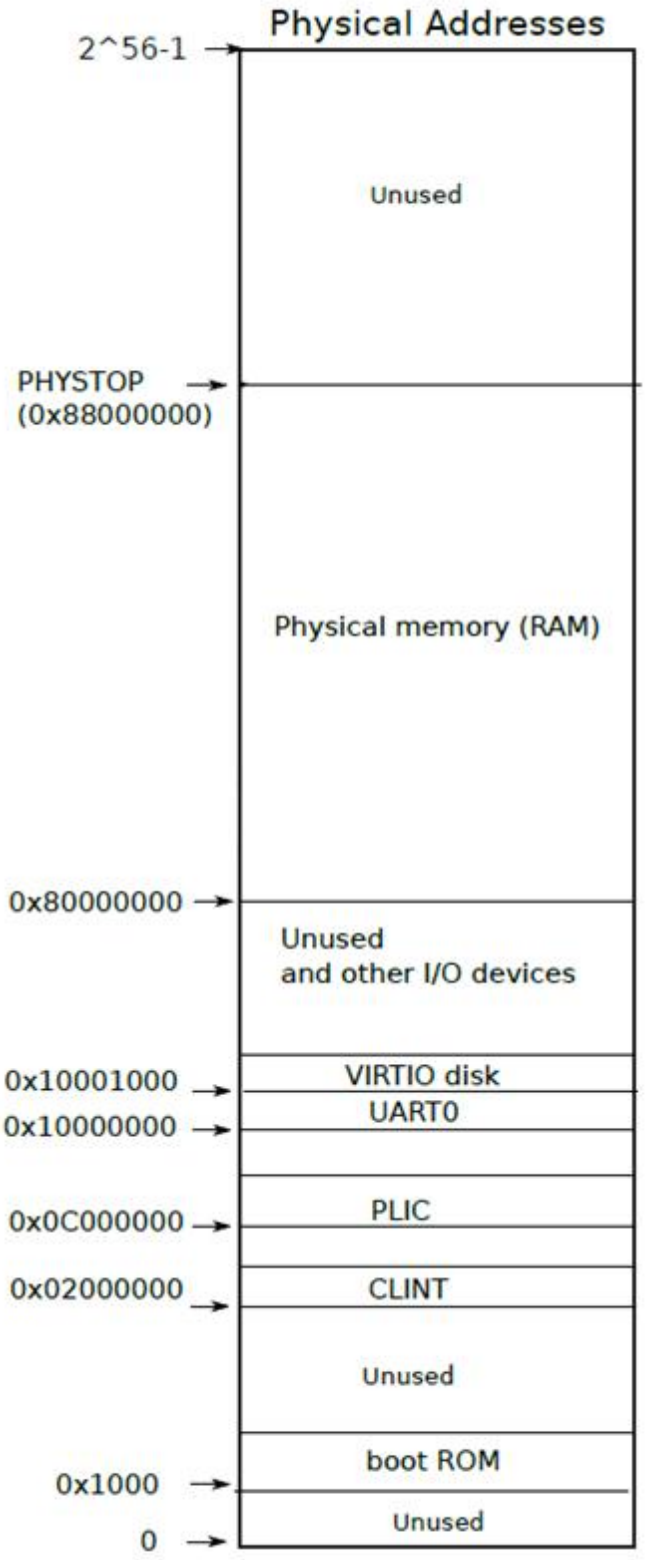


Vykonavanie programu

- Ako sa CPU dostane k dalsej instrukcii?
  - Specialny register CPU, ktory sa zvysuje po kazdej instrukcii
  - Automaticky modifikovany niektorymi instrukciami (volanie procedury, navrat z procedury, skok)



Fyzicke adresy: organizacia RISC-V



prevzate a upravene z

https://pdos.csail.mit.edu/6.828/2020/xv6/book-riscv-rev1.pdf

Opakovanie

- Dve hw technologie umoznujuce izolaciu procesov od jadra a vzajomne medzi sebou

1. Systemove volania (umoznuju procesom presne definovanym sposobom vykonat obsluhu sluzby v jadre OS)
2. Virtualna pamat (vytvara zdanie vlastnej pamate pre proces; znemoznuje zasahovat do pamate ineho procesu alebo jadra)

Pamat

- Majme v pamati umiestnene procesy (a kod OS) (obrazok)
- Nech sa v nejakej aplikacii nachadza chyba, ktora sposobuje obcasne zapisanie na nahodnu adresu v RAM-ke
- Ako zabezpecime, aby sa neprepisali udaje/kod nejakej aplikacie alebo jadra OS?

Adresny priestor

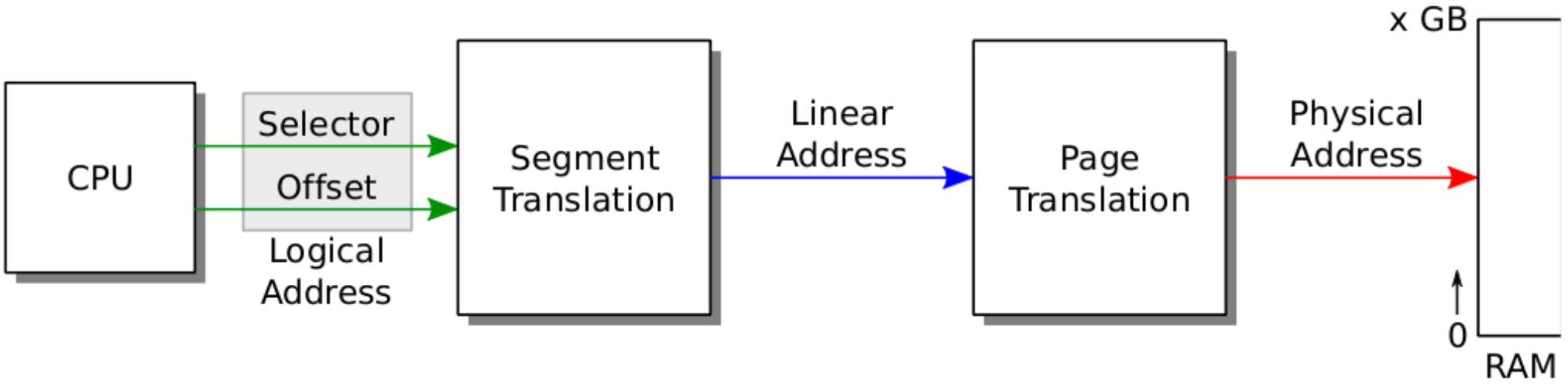
- Izolaciou pristupu procesu k pamatovemu priestoru
- Kazdy proces bude mat “vlastnu” pamat
- Co to znamena?
  - Moze do nej zapisovat a citat z nej
  - Nemoze zapisovat a citat z pamate ineho procesu alebo jadra OS (vid obrazok)
- Otazka: ako vtesnat (multiplexovat) viacero takychto samostatnych pamatovych boxov do jednej RAM-ky a zachovat izolaciu medzi nimi?

Adresny priestor

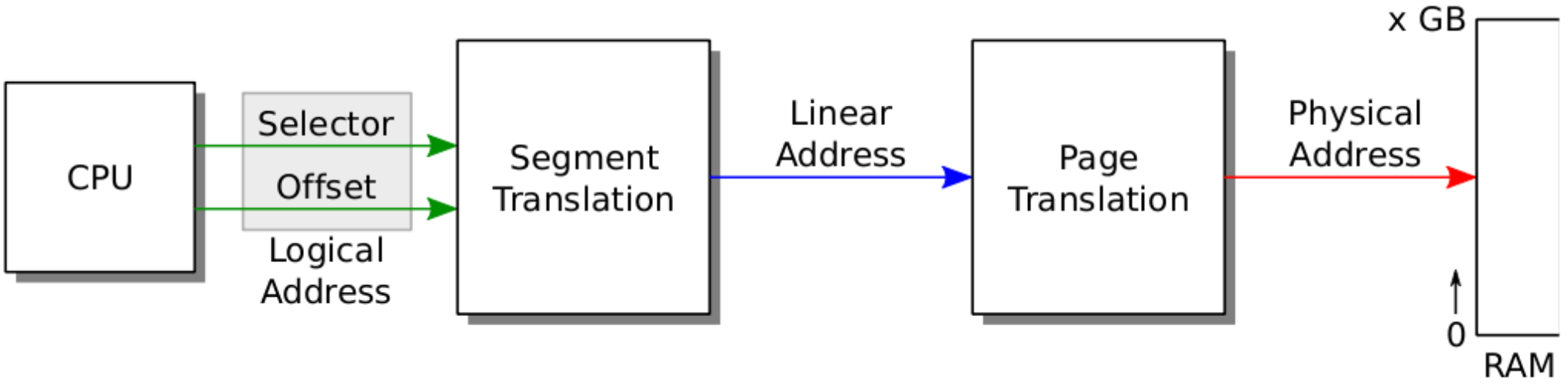
- Jestvuju viacere moznosti, najrozsirenejsie su

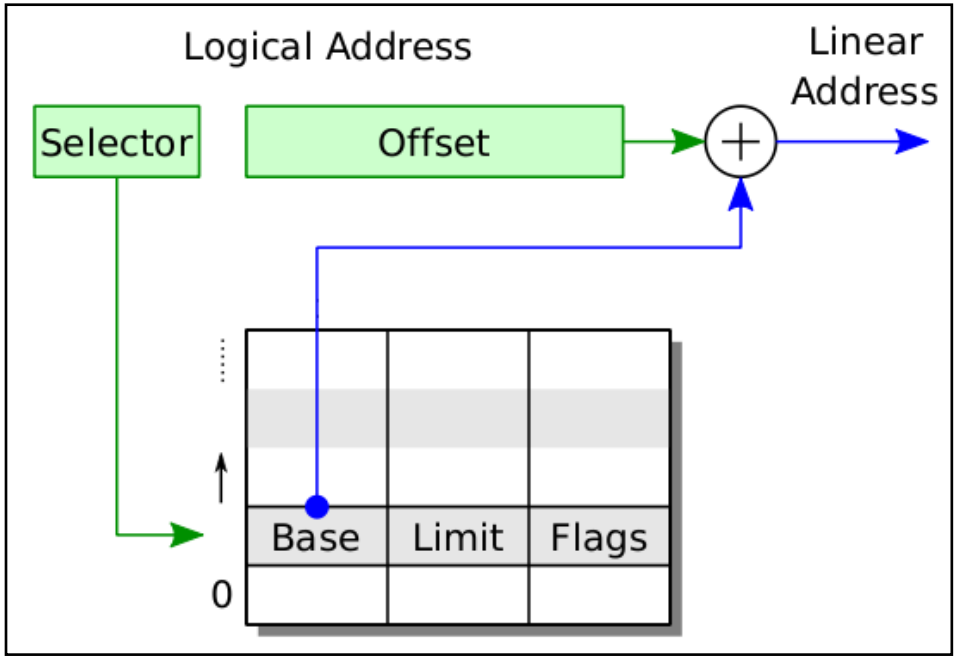
1. Segmentacia
2. Strankovanie
3. Kombinacia seg+str alebo str+seg

Schema prekladu adresy

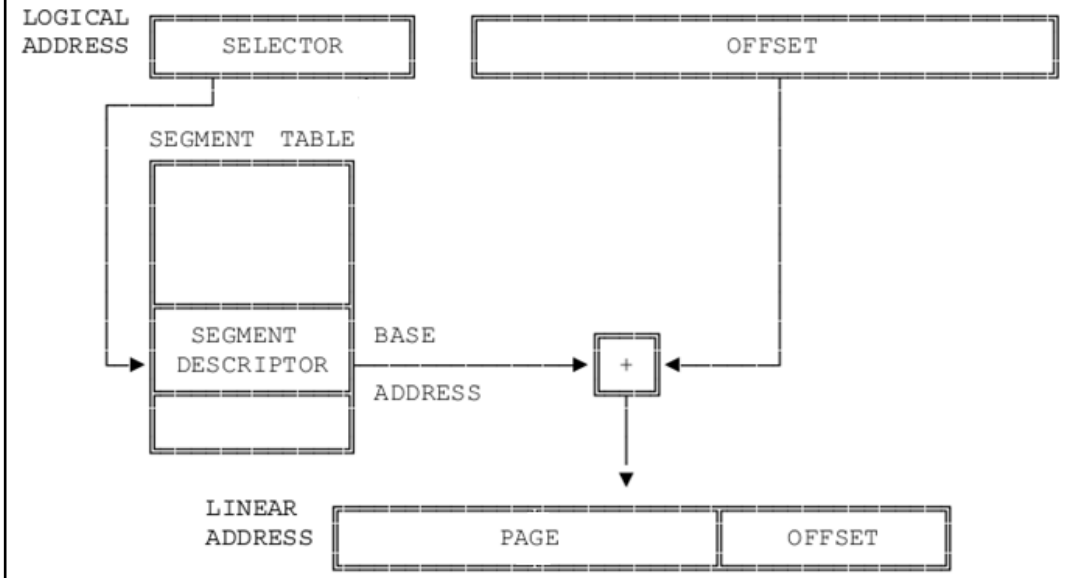


Schema prekladu adresy





Prevzate a upravene z INTEL 80386 PROGRAMMER'S REFERENCE MANUAL 1986



Segmentacia

Zhrnutie

- CPU pouziva Logicke (Virtualne) Adresy, VZDY!
- Vysledok segmentacie je preklad Logickej (Virtualnej) Adresy na tzv. Linearnu Adresu

- Preco linearnu?

Strankovanie

- Ide o mechanizmus nepriamej adresacie
- Zjednoduseny nacrt

- Odkaz na aktualnu tabulku stranok (tabulku, ktora sa pri preklade pouziva), sa nachadza v specialnom registri CPU
- Menit hodnotu tohto CPU je mozne iba v privilegovanom rezime (kernel mod)

Strankovanie

- CPU startuje v rezime, kedy je strankovanie vypnute
  - Preco? Ak by bolo zapnute, tak mame problem sliepky a vajca…
- Inicializacny kod jadra
  - Vyplni tabulku
  - Nastavi register, ktory ukazuje na tabulku stranok
  - Zapne strankovanie CPU

Strankovanie

- Jadro OS v privilegovanom rezime moze menit obsah registra, ktory ukazuje na tabulku stranok
- Teda iba jadro OS moze menit mapovanie, ktora fyzicka pamat je dostupna z ktorej virtualnej adresy (ci uz samotneho jadra alebo procesu)
- Uzivatelsky proces bezi v user mode CPU, nemoze menit obsah tohto registra – pokus o jeho zmenu vyvola vynimocny stav

Strankovanie

- Kazda tabulka stranok definuje vlastne mapovanie, vlastny adresny priestor, ktory nazyvame VIRTUALNY
- Obrazok s mapovanim procesov do RAM

Strankovanie

- Strankovanie umoznuje (podobne ako segmentacia, ale nespominali sme)
  - Rozlisenie opravnenia typu pristupu (Read / Write)
  - Urcit, ci mapovanie jestvuje (Present)
  - Rozlisenie opravnenia pristupu (Kernel / User)
  - Zistit, ci pristup k urcitej adrese vyvolal zmenu hodnot v pamati (Dirty)
  - Zistit, ci bolo vobec niekedy pristupene k nejakej adrese (A – access bit)
- Triky s virtualnom pamatou

Strankovanie – co nam k tomu treba

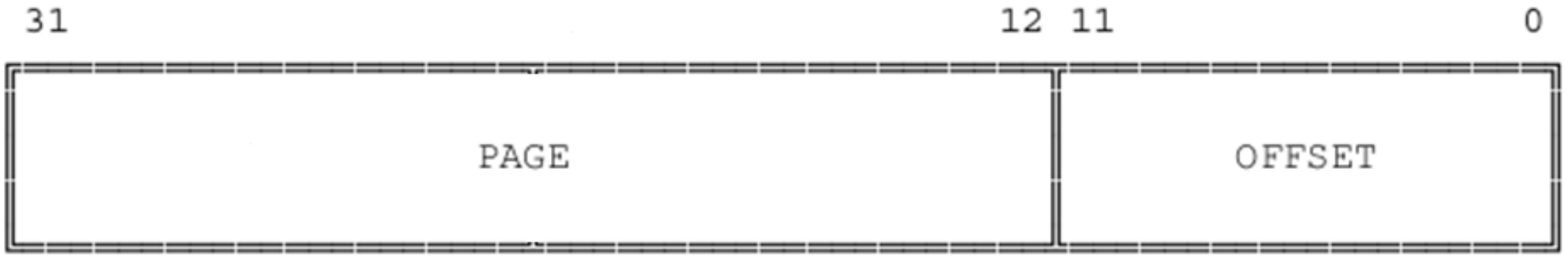
- Ramec (page frame) versus Stranka (page)

- Linearna adresa
- Tabulka stranok
- Polozka tabulky stranok

- Ako sa robi preklad

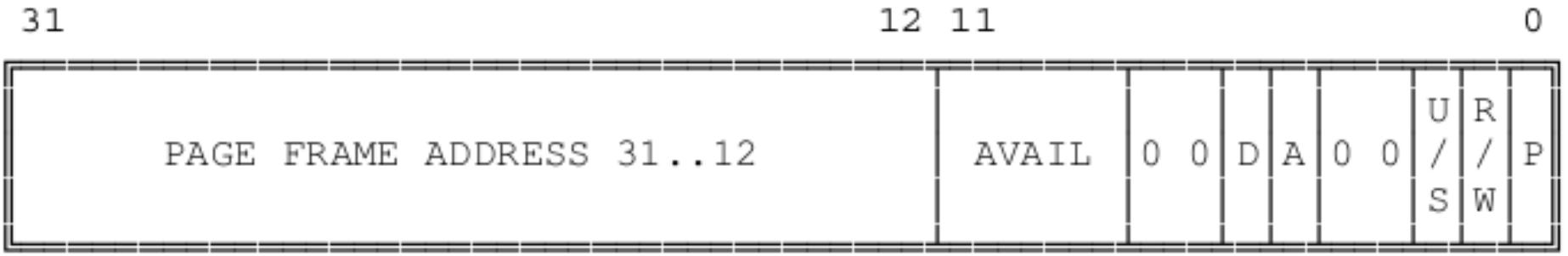
Prevzate a upravene z INTEL 80386 PROGRAMMER'S REFERENCE MANUAL 1986

Format linearnej adresy



Prevzate z INTEL 80386 PROGRAMMER'S REFERENCE MANUAL 1986

Polozka tabulky stranok (Intel)

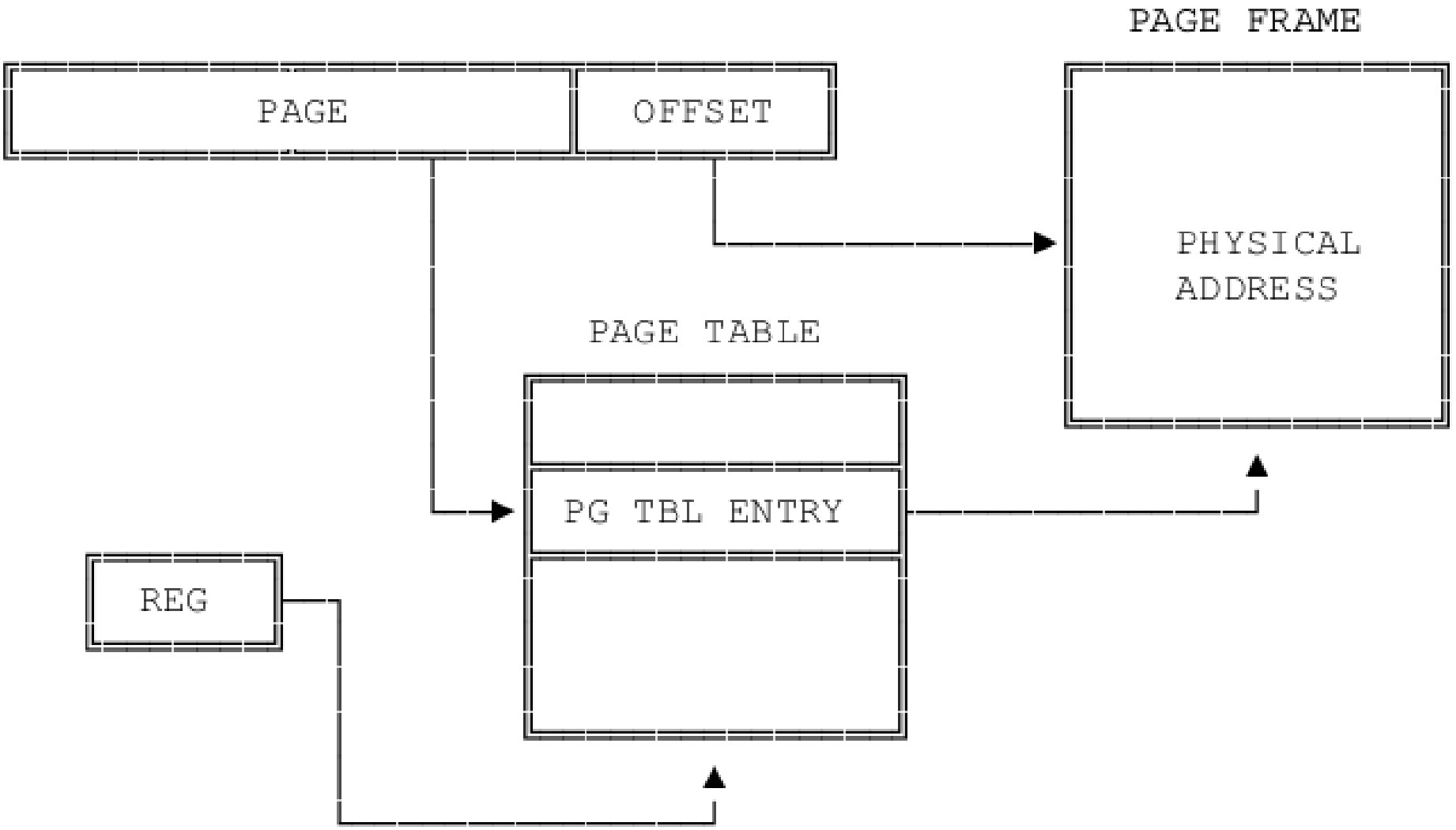


P - PRESENT
R/W - READ/WRITE
U/S - USER/SUPERVISOR
D - DIRTY
AVAIL - AVAILABLE FOR SYSTEMS PROGRAMMER USE

NOTE: 0 INDICATES INTEL RESERVED. DO NOT DEFINE.

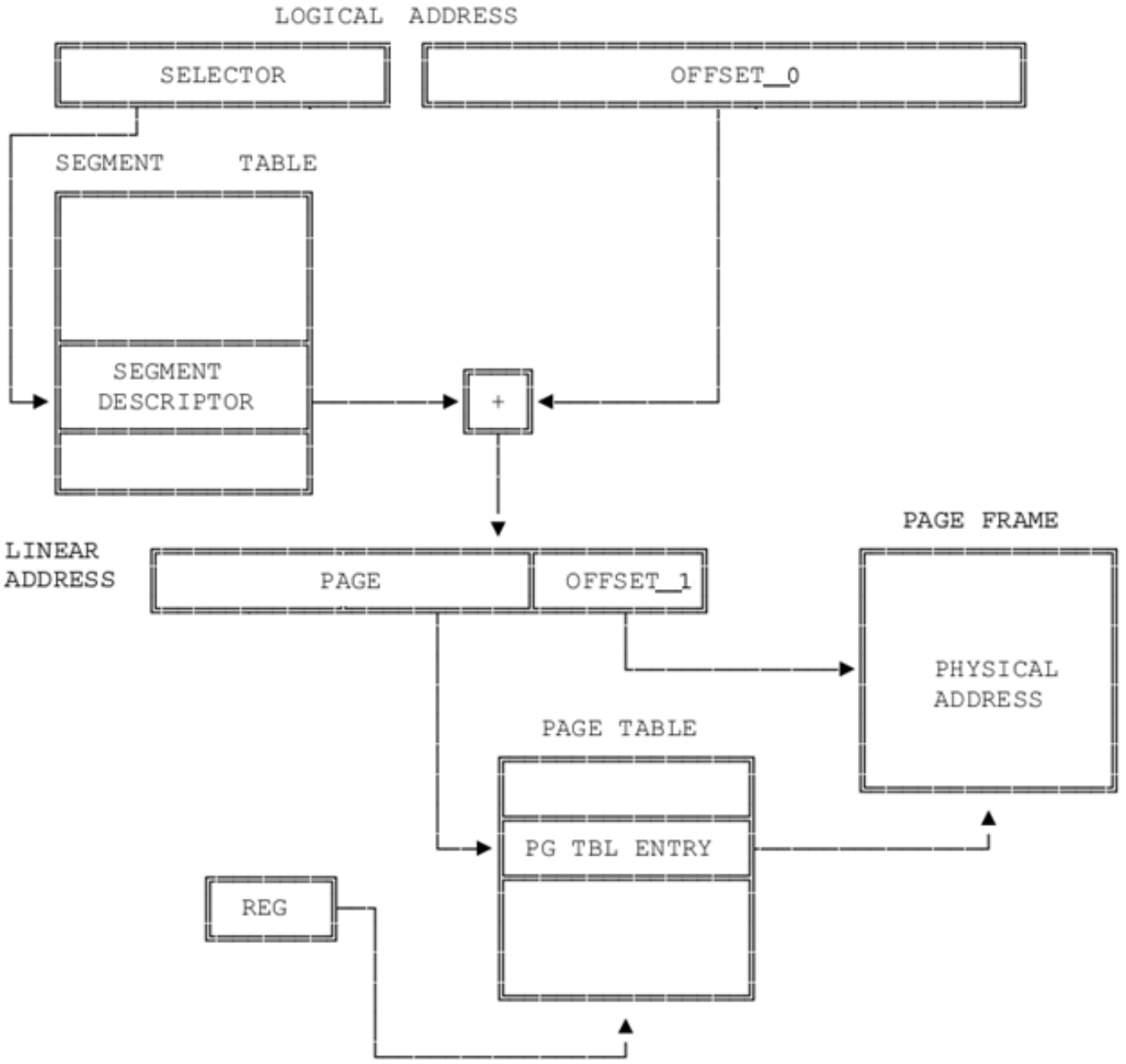
Prevzate a upravene z INTEL 80386 PROGRAMMER'S REFERENCE MANUAL 1986

Preklad Linearnej Adresy na Fyzicku



Prevzate a upravene z INTEL 80386 PROGRAMMER'S REFERENCE MANUAL 1986

Kombinacia Seg a Str



Strankovanie…

- Aby svet spel k lepsiemu, v takejto podobe (ako bolo na prednaske) sa strankovanie nikde nepouziva

- Viac o strankovani a virtualnej pamati na buducej prednaske

Domaca uloha

- Precitat kapitolu 2
- Operating system organization

- https://pdos.csail.mit.edu/6.828/2020/xv6/book-riscv-rev1.pdf

Online

- Konci Gmeet

- Zaciname pouzivat (od 1.10. vratane) live stream YouTube
- Odporucame zaregistrovat odber, aby ste nezmeskali prenos

MIT ;)